Fundamentos da Engenharia de Software

Conteudista: Prof. Me. Artur Marques
Revisão Textual: Prof. Me. Luciano Vieira Francisco

Objetivo da Unidade:

- Conhecer os fundamentos da engenharia de *software* e suas implicações na contemporaneidade.

Material Teórico

Material Complementar

Referências

Material Teórico

Introdução

Situaremos a engenharia antes de discorrer sobre engenharia de *software*, até mesmo para segregá-la e dar o devido foco.

A engenharia de forma geral é bastante antiga, não há uma data específica no tempo que possa ser confiável o suficiente para sua determinação e nem quem a inventou. Todavia, podemos remontar às descobertas e aos eventos arqueológicos que colocam certos personagens egípcios como os primeiros a deixarem registros de suas criações, foi o caso, por exemplo, de *Imhotep*, que foi o primeiro engenheiro registrado na história. Arquimedes seria outro dos mais conhecidos em criar – comumente armas.

Portanto, quando tratamos de engenharia pensamos imediatamente na Ciência que cria, transforma os recursos existentes em algo útil para a humanidade, ou ao menos nos tempos antigos para o clã, a aldeia ou o reino onde é utilizada.

De tal forma que quando pensamos no engenheiro, aquele que se vale do conhecimento da engenharia, temos algumas definições como, por exemplo: "Uma pessoa cujo trabalho é projetar ou construir máquinas, motores ou equipamentos elétricos, ou coisas como estradas, ferrovias ou pontes, usando princípios científicos" (CAMBRIDGE DICTIONARY, 2022).

E aí começam as grandes discussões quando olhamos sob a perspectiva da engenharia de *software*.

Apesar de no Brasil conselhos de classe terem colocado em seus portfólios a disponibilidade de carteiras colocando as carreiras de engenharia de *software* como "engenharia",

desmistificaremos algumas coisas no texto de um dos expoentes da área:

“A engenharia-como-disciplina: uma disciplina é uma maneira distinta de realizar uma atividade, uma combinação de conhecimento, habilidade e julgamento que deve ser aprendido. Qualquer ofício tem sua disciplina e nelas estão contidas diversas atividades que não são ofícios ou que são ofícios, como meditação ou calistenia. Nesse sentido, nem arquitetos, nem *designers* industriais, nem inventores de fim de semana são engenheiros. Arquitetura e *design* industrial têm uma disciplina facilmente distinta da engenharia. Os inventores de fim de semana não têm nenhuma disciplina; eles podem inventar da maneira que quiserem.

Engenharia de *software* não é engenharia nesse terceiro sentido. O corpo de conhecimento que os engenheiros devem aprender difere de maneiras importantes do corpo de conhecimento da engenharia de *software*.

Assim, por exemplo, os engenheiros têm que fazer cursos relacionados ao mundo material, como química e estatística; engenheiros de *software* não.

O corpo de conhecimento oficial da engenharia de *software* foi de fato um passo importante para esclarecer a distinção entre engenharia propriamente dita e engenharia de *software*. Requer que os engenheiros de *software* saibam coisas que outros engenheiros não sabem e não saibam algumas coisas que outros engenheiros sabem.

A engenharia como profissão: uma profissão é (podemos dizer) um número de indivíduos na mesma ocupação voluntariamente organizados para ganhar a vida servindo abertamente a um ideal moral de uma maneira moralmente permissível além do que a lei, o mercado, a moralidade e a opinião pública exigiriam.

Uma ocupação é uma disciplina pela qual alguém pode, e alguns o fazem, ganhar a vida. Tanto a engenharia quanto a engenharia de *software* são agora ocupações, mas, tendo (como acabamos de observar) diferentes disciplinas, devem ser ocupações diferentes. Essa é uma razão pela qual eles não podem compartilhar uma profissão.

Além disso, o Código de Ética e Prática Profissional da Engenharia de *Software* difere significativamente de todos os códigos de engenharia que conhecemos.

Os engenheiros de *software* devem, por exemplo, "moderar os interesses do engenheiro de *software*, do empregador, do cliente e dos usuários com o bem público". Os engenheiros que não são de *software*, não têm por hora esse dever de moderar.

A engenharia de *software* tornou-se, de fato, uma profissão.

O que não se tornou é parte da profissão de engenheiro.

Qualquer um que afirme o contrário deve encontrar um sentido de engenharia diferente daqueles distinguidos aqui, um que faça da engenharia de *software* uma parte da engenharia sem incluir também disciplinas, ocupações ou profissões, como arquitetura ou contabilidade, que claramente não fazem parte da engenharia."

- DAVIS, 2011, p. 32-34

Em linhas gerais, como aparente complicador para o engenheiro de *software* – e não necessariamente da disciplina de engenharia de *software* –, e que se ouve/lê muito sobre o tema da superautomação, entre outras coisas, é que um grande desafio seria o seguinte: o mundo pode não precisar de muitos engenheiros de *software* nos próximos anos. Portanto, pode-se

depender que a maior parte do que os engenheiros de *software* fazem atualmente, como codificação, será automatizada.

Todavia, a probabilidade de engenheiros se tornarem redundantes está longe de ser uma possibilidade remota. Há muitas outras áreas da própria engenharia de *software* que serão alavancadas e manterão a demanda alta de profissionais que atuem com engenharia de *software*.

A engenharia de *software* é um estudo detalhado desde o projeto, desenvolvimento e manutenção de *software*. Foi introduzida para resolver os problemas de projetos de *software* de baixa qualidade. A engenharia de *software* também surgiu para atender à imensa taxa de mudança nos requisitos do usuário e no ambiente no qual o aplicativo deveria funcionar.

A engenharia de *software* é um conceito em si, mas para entendê-la melhor você precisará saber o que cada parte do termo significa antes de entender completamente como operam juntas, vejamos:

- Engenharia é o desenvolvimento/criação/elaboração de produtos usando as melhores práticas, princípios e métodos;
- *Software* é uma coleção de códigos, documentos e acionadores que realizam um trabalho específico e atendem a um requisito específico.

Assim e em termos formais:

> “O IEEE, em seu padrão 610.12-1990, define engenharia de *software* como a aplicação de uma abordagem sistemática e disciplinada, que é uma abordagem computável para o desenvolvimento, operação e manutenção de *software*.

Fritz Bauer a definiu como “o estabelecimento e os princípios de engenharia padrão usados”. Ajuda-o a obter, de forma económica, *software* fiável e eficaz nas máquinas reais.

Boehm define a engenharia de *software*, que envolve “a aplicação prática do conhecimento científico ao *design* criativo e à construção de programas de computador”. Também inclui a documentação associada necessária para desenvolvê-los, operá-los e mantê-los.”

- MARTIN, 2022, p. 1

Histórico

A engenharia existe há séculos, senão milênios, tal como vimos. A engenharia de *software* – com este nome – fez a sua estreia em 1948 e não foi amplamente chamada assim até 1952. Nos últimos 50 anos, a engenharia de *software* fez uma viagem interessante e se tornou uma parte vital da formação e manutenção do mundo. Todavia, logo no início da década de 1960 um evento muito importante ocorreu e o efeito percebido é que apesar de tudo, muitos projetos de *software* ou falhavam ou eram cancelados, “estouravam” em tempo ou em orçamento, de modo que quanto maior era o *software*, mais tempo levava para construir, mais dinheiro era consumido e, por incrível que pareça, não era possível de se manter o sistema em nível funcional muitas das vezes.

Somando-se a isso, devido à velocidade que os mercados impuseram aos negócios na segunda metade do século XX, a concorrência entre empresas se acelerava, de modo que o advento das empresas multinacionais, a possibilidade de processamento de dados em nível comercial com equipamentos de grande e médio porte trouxeram impactos negativos na agilidade, flexibilidade e adaptação – exploradas fortemente por essas empresas.

Claro, você já deve estar imaginando que isso aumentou em muito a complexidade do hardware, o que por consequência exigia um *hardware* cada vez mais poderoso. Em suma, a chamada crise do *software* se formou principalmente porque a necessidade de se criar soluções em *software*

para atender a demandas como as da área de negócios era muito maior que a capacidade de se gerar *software* e sistemas de *software* novos.

Apesar de haver uma linha sendo desenvolvida para criar soluções – chamada, ainda de forma incipiente, de engenharia de *software* –, à época era desorganizada e não padronizada. De qualquer forma, havia motivadores suficientes em nível de eficiência, eficácia e negócios que exigiam a estruturação e fundamentação da engenharia de *software* para fazer frente a todas essas demandas.

Sobre os grandes aceleradores dessa crise de *software*, podem ser elencados, mas não exauridos, na seguinte lista:

- O custo de se possuir e manter o software era tão ou mais caro que desenvolver um novo;
- Os projetos eram executados por um tempo muito longo e inaceitável;
- O *software* construído, quando se conseguia entregar, era muito ineficiente;
- A qualidade do *software* era muito baixa;
- Rotineiramente, o *software* não atendia aos requisitos do usuário;
- Um projeto de média complexidade "estourava" o seu cronograma em, ao menos, 50%, rotineiramente;
- A quantidade de *softwares* não entregues formava a maioria dos projetos;
- Era muito difícil alterar, depurar e aprimorar um *software*.

Então, quando pensamos na resposta sobre a crise de *software*, consideramos, na engenharia de *software*, os seguintes desafios:

- *Softwares* grandes e complexos;
- Escalabilidade;
- Adaptabilidade;
- Custos;
- Natureza dinâmica do negócio representada no *software*;
- Gestão e controle da qualidade em geral.

Elementos Constituintes

A engenharia de *software* envolve a aplicação de princípios da Ciência da Computação, Ciência da Gestão e outros campos para o projeto e a construção de sistemas de *software*. A educação em engenharia de *software* é fundamentalmente diferente da educação em Ciência da Computação, Ciência da Administração ou outros campos constituintes, embora compartilhe uma grande área comum de preocupação.

À medida que avançamos em direção ao desenvolvimento de currículos coordenados de engenharia de *software*, torna-se obrigatório que identifiquemos princípios – e não apenas coleções aleatórias de técnicas sobre as quais construí-los.

Assim, são elementos da engenharia de *software*:

- Aspectos da Ciência da Gestão;
- Habilidades de comunicação;
- Resolução de problemas;
- Metodologia de *design*;

- Programação/codificação;
- Habilidades em gerência de configuração;
- Habilidades em domínio de negócios e domínio de problema.

São exemplos comumente utilizados para designar as motivações para lidar com a complexidade do *software* mediante o emprego da engenharia de *software*; sem se mencionar a própria arquitetura de *software*. Por exemplo, o *design* de *software* é dividido em três níveis: *design* de interface, *design* arquitetônico e *design* detalhado.

Isso abarcará a categoria de sistemas e, como abordaremos, a proposta de solução, por exemplo. Alguns projetos devem ser focados em confiabilidade, usabilidade e compreensão, enquanto outros projetos darão grande importância à testabilidade e eficiência

Segundo Pressman (2016), a engenharia de *software* possui três elementos constituintes principais, a saber:

- **Processos:** são utilizados para descrever uma sequência ou conjunto de passos que devem ser seguidos pelo time de engenharia ou desenvolvimento, levando-se em consideração que não existe uma única abordagem para um problema de construção de *software* pelo time de TIC devido à diversidade de tipos de sistemas e organizações que usarão essas soluções computacionais. Aqui entram os famosos paradigmas de desenvolvimento, tais como os tradicionais em cascata, em espiral, entre outros; como iterativo, incremental e os métodos ágeis, como o *Extreme Programming* (*XP*), *Scrum*, TDD, FDD, *Crystal*, DFDM etc., para o desenvolvimento de *software.*

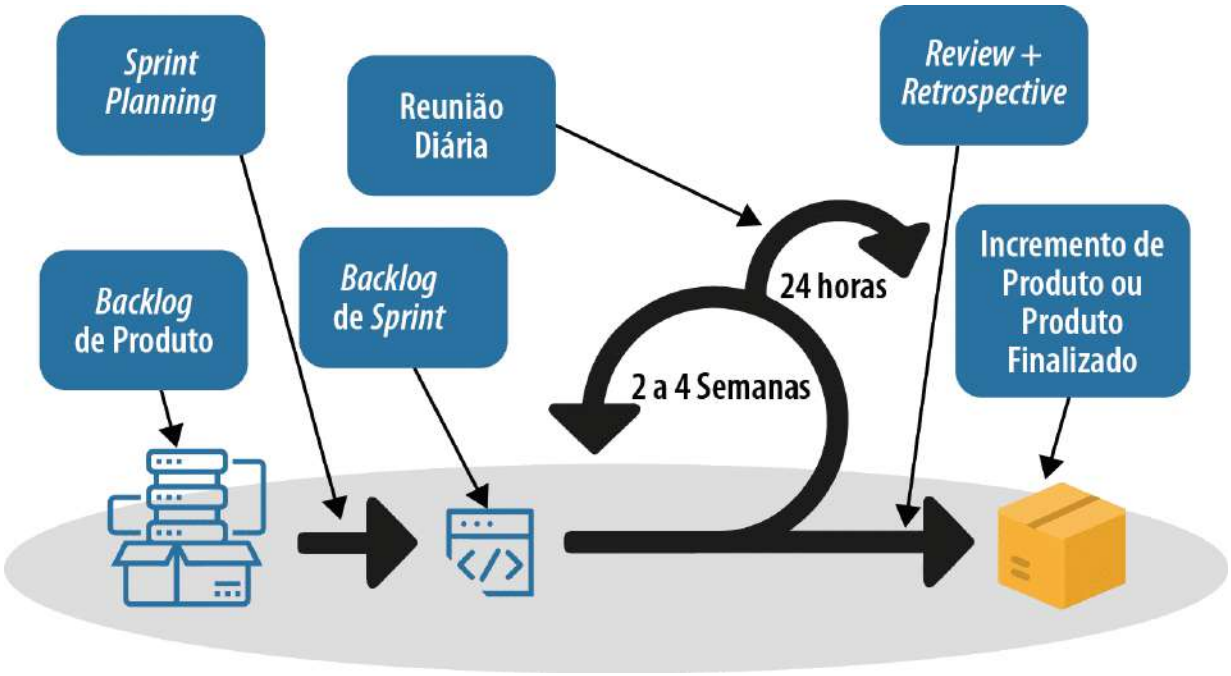

Figura 1 – Exemplo de um processo de desenvolvimento de *software* conhecido como *Scrum* (macrocomponentes)
Fonte: Adaptada de SIMAS *et al.*, s.d. | Flaticon

#ParaTodosVerem. Imagem do fluxo do processo *scrum* simplificado. Plano esquemático do trabalho envolvido num projeto de *software* típico utilizando *scrum*. Na extrema esquerda da figura há um bloco azul representando a primeira etapa, na qual as histórias dos usuários são reunidas e temos o *backlog* do produto. O próximo passo, representado por um bloco azul menor, contém apenas as histórias escolhidas para o time *scrum* trabalhar, chamamos isso de *sprint backlog*. Como terceiro item há dois ciclos, representados na cor verde por dois semicírculos, o primeiro dita a janela de tempo da duração de uma *sprint*: entre duas até quatro semanas. O segundo semicírculo representa a cerimônia *scrum* diária chamada *scrum daily meeting*. Por fim, com o último elemento mais à direita, há um pacote na cor marrom clara representando a entrega fruto do processo ocorrido na *sprint*. Fim da descrição.

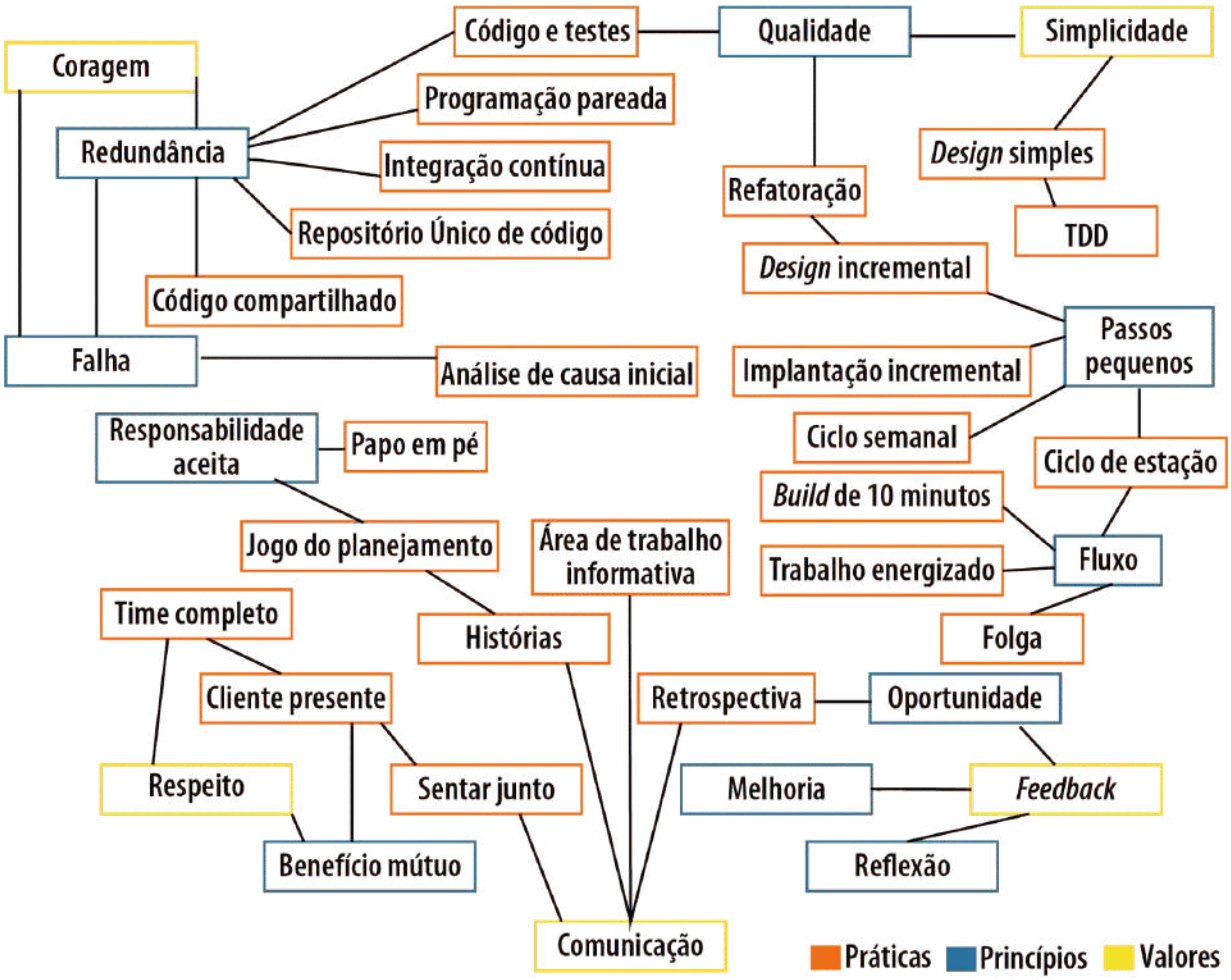

Figura 2 – Exemplo de processo de desenvolvimento ágil, com valores, princípios e práticas de XP

Fonte: Adaptada de aglicoop.org

#ParaTodosVerem. Imagem contendo os valores do *Extreme Programming*. Imagem colorida com vários balões de diálogo sendo composta uma nuvem. Os balões amarelos contêm os seguintes tópicos principais: comunicação, *feedback*, respeito, coragem e simplicidade. Os balões da cor azul são um nível de detalhamento dos balões amarelos com valores e benefícios complementares aos descritos nos balões amarelos: responsabilidade aceita, benefício mútuo, reflexão, melhoria, oportunidade, fluxo, passos pequenos, qualidade, redundância e falha. Por fim, nos balões vermelhos estão as atitudes esperadas e necessidade para a boa dinâmica: time completo, cliente presente, sentar junto, histórias, área de trabalho informativa, retrospectiva, folga, trabalho energizado, *build* de 10 minutos, estação, ciclo semanal, implantação incremental, *design* incremental, tdd, refatoração, *design* simples, código e testes, programação em pares, integração contínua, repositório único de código, código compartilhado, jogo de planejamento e reunião em pé. Fim da descrição.

- **Métodos:** técnicas ou procedimentos usados no desenvolvimento de *software*. É um conceito amplo que inclui várias fases de desenvolvimento de *software*, tais como *design*, desenvolvimento e teste. Isso também é conhecido como Ciclo de Vida de Desenvolvimento de Sistemas (SDLC). Portanto, é o passo a passo, por exemplo, comunicação, levantamento de requisitos, estudo de viabilidade, análise de sistema, projeto de *software*, codificação, teste, integração, entre outros.

Figura 3 – Exemplo de alto nível de um SDLC
Fonte: Adaptada de Freepik

#ParaTodosVerem. Exemplo de ciclo de vida do desenvolvimento de *software*. Círculo em tons de azul composto por setas sequenciadas, cada uma com um

componente de uma fase sequenciada do ciclo de vida. Apresentam a seguinte ordem crescente: Necessidade do cliente, especificação de requisitos, validação junto ao cliente, desenvolvimento de testes, entrega, homologação e implantação, por fim, utilização da solução e suporte. Fim da descrição.

- **Ferramentas:** as ferramentas de *software* são usadas para realizar e investigar os processos de negócio, documentar o processo de desenvolvimento do *software* e otimizar todos os processos. Ao se utilizar essas ferramentas no processo de desenvolvimento de *software*, o resultado dos projetos será mais produtivo. Tais ferramentas são também conhecidas como *Case – Computer-Aided Software Engineering*, ou engenharia de *software* assistida por computador. Compila um amplo conjunto de ferramentas de produtividade usadas no desenvolvimento de *software*; além disso, criam uma estrutura para o gerenciamento de projetos e se destinam a ajudar os usuários a se manterem organizados e a melhorar a produtividade.

Em linhas gerais, as ferramentas *Case* se dividem em três tipos:

- ***Upper-Case*:** apoiam etapas iniciais como iniciação, planejamento, análise e projeto;

- ***Lower-Case*:** apoiam a parte física/infra da criação do *software* e levam em conta implementação, testes e manutenção;

- ***Integrated-Case*:** cobrem todo o SDLC.

Por fim, não podemos nos esquecer que na engenharia de *software* há constante foco em produzir com qualidade. É uma parte crítica de um processo de desenvolvimento de *software* bem-sucedido. Quanto mais intensiva for a garantia de qualidade do *software*, melhor será o negócio a longo prazo.

Além de atender aos requisitos gerais do seu projeto, sua equipe de desenvolvimento deve atender a certos padrões de qualidade técnica, nos quais a indústria de desenvolvimento de *software* depende. Garantir que esses padrões sejam atendidos de maneira confortável e rígida é a garantia de qualidade de *software* implicada.

A garantia de qualidade de *software* ocorre antes, durante e após o ciclo de vida de desenvolvimento de *software*.

As organizações devem verificar se as características internas e externas de um produto de *software* estão à altura. As qualidades externas descrevem como o *software* funciona em tempo real, enquanto as qualidades internas referem-se aos blocos de construção mais fundamentais do *software*, como o código.

Existem duas abordagens principais para a garantia de qualidade de *software*, a abordagem de:

- Gerenciamento de defeitos;
- Atributos.

Preocupamo-nos de forma básica quanto à qualidade de *softwar*e com seis atributos, a saber:

- Funcionalidade;
- Confiabilidade;
- Usabilidade;

- Eficiência;
- Manutenibilidade;
- Portabilidade.

Ademais, há três tipos fundamentais de qualidade de *software* visando à satisfação dos clientes e usuários em geral, vejamos:

- **Qualidade básica:** todo produto ou serviço deve ter certo padrão mínimo e deve satisfazer aos requisitos básicos. No caso da indústria de tecnologia, o serviço que é fornecido, por exemplo *software*, deve estar livre de erros e redundâncias, deve ter uma lógica de fácil compreensão, deve ser facilmente reprogramável e fácil de compilar. Assim, quando um produto ou serviço atende aos critérios básicos, o nível de qualidade é denominado qualidade básica;
- **Qualidade de desempenho:** é a medida de sua robustez. No que diz respeito a um serviço, por exemplo *software*, a qualidade do desempenho exige que ele não falhe ou falhe e funcione diligentemente, apesar das entradas desordenadas. Isso exige que o *software* tenha robustez no que diz respeito à execução das instruções de entrada;
- **Qualidade de excitação:** é o nível em que o produto ou serviço não apenas apresenta o melhor desempenho, mas também oferece algo mais do que a expectativa que excita os clientes. Para serviços pode-se incluir ajuda para a solução de problemas, fornecida a qualquer momento, atualização periódica sem ser solicitada, experiência e confiabilidade robustas e sem erros. Assim, quando algo mais do que o esperado é recebido, o cliente fica naturalmente interessado em obter os mesmos serviços no futuro e definitivamente inspira outros clientes em potencial.

Figura 4 – Elementos da engenharia de *software*

#ParaTodosVerem. Imagem contendo macro elementos referentes à engenharia de *software*. Imagem em escala de azul construída através de elipses concêntricas, sendo a de nível mais baixo, descrita como: Foco na qualidade. Na elipse superior, ligeiramente menor, descrita como: Processo. Na terceira elipse, a descrição: métodos. Na quarta e última elipse, estão os dizeres: ferramentas. Fim da descrição.

Site

Swebok Version 3

Acima de tudo há um documento da engenharia de *software*, intitulado *Swebok* V3.0, completamente revisado e atualizado, conhecido como o "*Guide to the Software Engineering Body of Knowledge*", ou Guia para o Corpo de Conhecimento em Engenharia de *Software*, sendo respeitado internacionalmente.

Clique no botão para conferir o conteúdo.

ACESSE

Importância e Desafios

Vejamos alguns dos desafios que a engenharia de *software* enfrenta atualmente:

- Em áreas críticas de saúde, segurança, espaço, aviação, usinas nucleares, indústria da guerra etc., o custo da falha de *software* pode ser enorme porque vidas estão em risco;
- Aumento da demanda do mercado por tempos de resposta mais rápidos;
- Lidar com a crescente complexidade do *software* precisa de novos e melhores aplicativos;
- A diversidade de sistemas de *software* deve estar se comunicando entre si, principalmente agora, em que estamos sujeitos a *Internet of Things* (*IoT*), Inteligência Artificial (IA) e seus derivados; de forma geral, diversos *softwares* embarcados que conversam ou utilizam dados de sistemas legados, sistemas transacionais baseados em nuvem, bancos distribuídos ou novas formas autônomas de sistemas.

O Engenheiro de *Software* e suas Atribuições

Engenheiros de *software* de todos os tipos, funcionários em tempo integral, fornecedores, trabalhadores contratados ou trabalhadores em meio período são membros importantes da comunidade de Tecnologia da Informação (TI).

Ademais, o que os engenheiros de *software* fazem? Aplicam os princípios da engenharia de *software* ao projeto, desenvolvimento, à manutenção, ao teste e à avaliação de *software*. Há ampla discussão sobre o grau de educação e/ou certificação que deve ser exigido para engenheiros de *software*.

Os engenheiros de *software* são peritos no processo de desenvolvimento de *software*, embora comumente precisem de informações do líder de TI sobre os requisitos de *software* e qual deve ser o resultado. Independentemente da educação formal, todos os engenheiros de *software* devem trabalhar dentro de um conjunto específico de melhores práticas de engenharia de *software* para que outros possam fazer parte desse trabalho ao mesmo tempo.

A engenharia de *software* quase sempre inclui grande quantidade de trabalho em equipe. *Designers*, escritores, codificadores, testadores, vários membros do time e toda a equipe de TI precisam entender o código.

Os engenheiros de *software* devem entender como trabalhar com várias linguagens comuns de computador, incluindo *NET*, *Python*, PHP, *Java*, C# e C++, sendo que *JavaScrip*t é a linguagem de programação mais usada. *Python* subiu no *ranking*, superando C#, assim como superou PHP.

Em engenharia de *software* temos os chamados *hard* e *soft skills* e podemos definir como sendo os atributos pessoais que apoiam a consciência situacional e aumentam a capacidade de um indivíduo em realizar um trabalho, ou ainda como sinônimo de habilidades pessoais ou inteligência emocional.

Soft skills permitem que alguém interaja de forma eficaz e harmoniosa com outras pessoas.

Eis importantes aspectos para a engenharia de *software*:

- Habilidades de resolução de problemas e pensamento crítico;
- Habilidades de comunicação;
- Pessoas e habilidades interpessoais;
- Autoconsciência;
- Autoaprendizagem;

- Responsabilidade;
- Gerenciamento de tempo;
- Inteligência emocional.

Por outro lado, *hard skills* são habilidades específicas, ou capacidades que um indivíduo pode possuir e demonstrar de forma medida. Possuir uma *hard skill* conota maestria e experiência do indivíduo para realizar uma tarefa específica ou uma série de tarefas para concluir um trabalho. São habilidades técnicas como conhecer e saber aplicar um SDLC ou um paradigma de desenvolvimento, elicitar um requisito, conhecer e desenvolver uma arquitetura de sistemas, saber sobre a gerência de domínio ou gerência de configuração, por exemplo.

Tendências Atuais

Como os produtos digitais determinam a criação de negócios sustentáveis e adaptáveis, desenvolvedores de *software* e especialistas em TI, através da engenharia de *software* e das melhores práticas, desempenham papel fundamental na construção do mundo pós-pandemia de *Coronavirus Disease* 2019 (Covid-19). Vejamos as ferramentas e estratégias utilizadas para se alcançar resultados melhores na atualidade, que você precisa conhecer e com o tempo dominar para se manter competitivo(a) no mercado de trabalho:

- Revisões de código automatizadas, por exemplo, *Veracode*, *CodeClimate*, *Codacy*, *DeepSource*;
- Foco em padrões de qualidade de *software*, por exemplo, ISO/IEC 27001;
- Migração de sistemas legados, incluindo a construção de API e desafios, por exemplo, desempenho ruim, altos custos de manutenção, incapacidade de atender aos requisitos do cliente, violações de segurança de dados etc.;
- *Kubernetes*, *dockers*, *microservices* e SOA, por exemplo, conteinerização e uma arquitetura de microsserviços baseada em nuvem;

- Prevalência de tecnologia nativa em nuvem, por exemplo, produtos e serviços de *software* hospedados em nuvens públicas, privadas ou híbridas; são independentes de infraestrutura e podem ser executados em vários servidores, a fim de que qualquer empresa possa adotá-los facilmente;

- *DevOpsSecBiz*, juntar todos os ambientes que impactam negócios para incrementar a agregação de valor e velocidade de integração;

- *Blockchain* para além de *criptomoedas*;

- Expansão de aprendizado de máquina com GPT3, AutoML e IA sem código, por exemplo, sistemas de recomendações e anúncios personalizados, reconhecimento de imagem e voz, *chatbots* e assistentes virtuais, sistemas de *software* de detecção de fraude, carros autônomos e análises preditivas, entre outros;

- IA restrita, por exemplo, reconhecimento de imagem, sistemas de reconhecimento de fala, sistemas de *software* de recomendação personalizada;

- Ferramentas de observabilidade do *DevOps*;

- Gerenciamento de infraestrutura com *GitOps*, por exemplo, prática para a automação de infraestrutura e sua integração;

- *Python* em primeiro lugar, depois *Java* – apesar de todo o legado no mundo – é a ordem de linguagens favoritas dos desenvolvedores que criam soluções de *software* com recursos baseados em inteligência artificial e *machine learning*;

- *Low-code/no-code* para acelerar o desenvolvimento de *software* e melhorar a produtividade;

- *Site Reliability Engineering* (SRE) e *DevOps* trabalhando juntos;

- Banco de dados multimodal e multifuncional, por exemplo, o *Azure CosmosDB*;

- *CyberSecurity*;

- *TensorFlow* para *deep learning*;

- Principais estruturas de desenvolvimento web como *react* e *angular*, por exemplo, *react* com suporte do *Facebook* garante flexibilidade e facilidade de *Search Engine Automation* (*SEO*) para as suas soluções de *software*. Com *Angular*, do *Google*, podemos criar aplicativos nativos e multiplataforma rápidos e de alto desempenho;

- **PWA:** aplicativos da *Web* Progressivos;

- **TCoE:** expansão dos Centros de Excelência de Testes;

- Demanda crescente por terceirização, por exemplo, no desenvolvimento de *software* mais rápido, eficiência de custos, melhor qualidade, redução de riscos e recursos liberados da empresa;

- Abordagens diferenciadas para o trabalho remoto e híbrido são tendências sem retorno para o mercado de Tecnologias de Informação e Comunicação (TIC) mundial; por exemplo, novos padrões de gerenciamento de equipe, comunicação e colaboração, a fim de que os funcionários possam realizar as suas tarefas sem entrar em um escritório físico.

Código de Ética da Engenharia de *Software*

O Código de Ética da Engenharia de *Software* se concentra no fornecimento de *software* de alta qualidade. Compromete-se em analisar, especificar, projetar, desenvolver, testar e manter *software* que seja benéfico e eficaz para a empresa ou para o cliente.

Os objetivos do Código de Ética são os seguintes:

- Definir o encargo da associação profissional;

- Delinear os seus poderes em relação a membros e não membros;

- Descrever, mediante declarações gerais sobre princípios de conduta ética, para preparar engenheiros de *software* ao cumprimento de seu dever junto ao público, à profissão e aos colegas.

Vejamos a seguir uma síntese do Código de Ética:

- **Público:** os engenheiros de *software* devem atuar de forma consistente com o interesse público;

- **Cliente e empregador:** os engenheiros de *software* devem agir de acordo com o melhor interesse de seu cliente e empregador, de modo consistente com o interesse público;

- **Produto:** os engenheiros de *software* devem garantir que os seus produtos e as modificações relacionadas atendam aos mais altos padrões profissionais possíveis;

- **Julgamento:** os engenheiros de *software* devem manter a integridade e independência em seu julgamento profissional;

- **Gestão:** os gerentes e líderes de engenharia de *software* devem aderir e promover uma abordagem ética na gestão do desenvolvimento e da manutenção de *software*;

- **Profissão:** os engenheiros de *software* devem promover a integridade e reputação da profissão de acordo com o interesse público;

- **Colegas:** cabe aos engenheiros de *software* ser justos e solidários com os seus colegas;

- **Eu:** os engenheiros de *software* devem participar na aprendizagem ao longo da vida sobre o exercício de sua profissão, assim como promover uma abordagem ética no exercício da profissão.

Vamos ao texto completo do Código de Ética da Engenharia de *Software*, de acordo com a *Association for Computer Machinery* (1997, p. 110-118):

“**Princípio 1 – Público**

Os engenheiros de *software* devem agir de forma consistente com o interesse público. Em particular, os engenheiros de *software* devem, conforme apropriado:

1.01 – Aceitar total responsabilidade por seu próprio trabalho.

1.02 – Moderar os interesses do engenheiro de *software*, do empregador, do cliente e dos usuários com o bem público.

1.03 – Aprove o *software* somente se eles tiverem uma crença bem fundamentada de que é seguro, atende às especificações, passa nos testes apropriados e não diminui a qualidade de vida, diminui a privacidade ou prejudica o meio ambiente. O efeito final do trabalho deve ser para o bem público.

1.04 – Divulgar às pessoas ou autoridades apropriadas qualquer perigo real ou potencial para o usuário, o público ou o meio ambiente, que eles razoavelmente acreditem estar associado ao *software* ou documentos relacionados.

1.05 – Cooperar nos esforços para tratar de assuntos de grande interesse público causados pelo *software*, sua instalação, manutenção, suporte ou documentação.

1.06 – Seja justo e evite enganos em todas as declarações, especialmente as públicas, sobre *software* ou documentos, métodos e ferramentas relacionados.

1.07 – Considere questões de deficiência física, alocação de recursos, desvantagem econômica e outros fatores que podem diminuir o acesso aos benefícios do *software*.

1.08 – Ser incentivado a oferecer habilidades profissionais para boas causas e contribuir para a educação pública sobre a disciplina.

Princípio 2 – Cliente e empregador

Os engenheiros de *software* devem agir de acordo com os melhores interesses de seu cliente e empregador, de acordo com o interesse público. Em particular, os engenheiros de *software* devem, conforme apropriado:

2.01 – Prestar serviço em suas áreas de competência, sendo honesto e direto sobre quaisquer limitações de sua experiência e formação.

2.02 – Não usar conscientemente *software* obtido ou retido de forma ilegal ou antiética.

2.03 – Usar a propriedade de um cliente ou empregador somente de maneiras devidamente autorizadas e com o conhecimento e consentimento do cliente ou empregador.

2.04 – Certifique-se de que qualquer documento no qual eles se baseiam tenha sido aprovado, quando necessário, por alguém autorizado a aprová-lo.

2.05 – Manter em sigilo todas as informações confidenciais obtidas em seu trabalho profissional, quando tal confidencialidade for compatível com o interesse público e compatível com a lei.

2.06 – Identifique, documente, colete evidências e relate imediatamente ao cliente ou ao empregador se, na opinião deles, um projeto for suscetível de falhar, se provar muito caro, violar a lei de propriedade intelectual ou ser problemático.

2.07 – Identificar, documentar e relatar questões significativas de interesse social, das quais eles tenham conhecimento, em *software* ou documentos relacionados, ao empregador ou ao cliente.

2.08 – Não aceite nenhum trabalho externo prejudicial ao trabalho que eles realizam para seu empregador principal.

2.09 – Não promover nenhum interesse adverso ao seu empregador ou cliente, a menos que uma preocupação ética superior esteja sendo comprometida; nesse caso, informe o empregador ou outra autoridade competente sobre a preocupação ética.

Princípio 3 – Produto

Os engenheiros de *software* devem garantir que seus produtos e modificações relacionadas atendam aos mais altos padrões profissionais possíveis. Em particular, os engenheiros de *software* devem, conforme apropriado:

3.01 – Esforce-se por alta qualidade, custo aceitável e um cronograma razoável, garantindo que as compensações significativas sejam claras e aceitas pelo empregador e pelo cliente e estejam disponíveis para consideração do usuário e do público.

3.02 – Garantir metas e objetivos adequados e alcançáveis para qualquer projeto em que trabalhem ou proponham.

3.03 – Identificar, definir e abordar questões éticas, econômicas, culturais, legais e ambientais relacionadas aos projetos de trabalho.

3.04 – Garantir que eles sejam qualificados para qualquer projeto em que trabalhem ou se proponham a trabalhar por uma combinação adequada de educação e treinamento e experiência.

3.05 – Certifique-se de que um método apropriado seja usado para qualquer projeto em que eles trabalhem ou se proponham a trabalhar.

3.06 – Trabalhar para seguir os padrões profissionais, quando disponíveis, que sejam mais adequados para a tarefa em questão, partindo deles apenas quando for ética ou tecnicamente justificado.

3.07 – Esforce-se para entender completamente as especificações do *software* em que eles trabalham.

3.08 – Certifique-se de que as especificações do *software* em que eles trabalham foram bem documentadas, satisfaçam os requisitos dos usuários e tenham as aprovações apropriadas.

3.09 – Garanta estimativas quantitativas realistas de custo, cronograma, pessoal, qualidade e resultados em qualquer projeto em que trabalhem ou se proponham a trabalhar e forneçam uma avaliação de incerteza dessas estimativas.

3.10 – Garantir testes adequados, depuração e revisão de *software* e documentos relacionados nos quais eles trabalham.

3.11 – Garantir a documentação adequada, incluindo problemas significativos descobertos e soluções adotadas, para qualquer projeto em que trabalhem.

3.12 – Trabalhar para desenvolver *softwares* e documentos relacionados que respeitem a privacidade daqueles que serão afetados por esse *software*.

3.13 – Tenha o cuidado de usar apenas dados precisos obtidos por meios éticos e legais e use-os apenas de maneiras devidamente autorizadas.

3.14 – Manter a integridade dos dados, sendo sensível a ocorrências desatualizadas ou falhas.

3.15 – Trate todas as formas de manutenção de *software* com o mesmo profissionalismo de um novo desenvolvimento.

Princípio 4 – Julgamento

Os engenheiros de *software* devem manter a integridade e independência em seu julgamento profissional. Em particular, os engenheiros de *software* devem, conforme apropriado:

4.01 – Temperar todos os julgamentos técnicos pela necessidade de apoiar e manter os valores humanos.

4.02 – Somente endossar documentos elaborados sob sua supervisão ou dentro de suas áreas de competência e com os quais estejam de acordo.

4.03 – Manter a objetividade profissional em relação a qualquer *software* ou documentos relacionados que sejam solicitados a avaliar.

4.04 – Não se envolver em práticas financeiras enganosas, como suborno, cobrança dupla ou outras práticas financeiras impróprias.

4.05 – Divulgue a todas as partes interessadas os conflitos de interesse que não podem ser evitados ou escapados de forma razoável.

4.06 – Recusar-se a participar, como membros ou conselheiros, em um órgão privado, governamental ou profissional preocupado com questões relacionadas a *software*, em que eles, seus empregadores ou seus clientes tenham potenciais conflitos de interesse não revelados.

Princípio 5 – Gestão

Os gerentes e líderes de engenharia de *software* devem aderir e promover uma abordagem ética para o gerenciamento de desenvolvimento e manutenção de *software*. Em particular, os engenheiros de *software* gerentes ou líderes devem, conforme apropriado:

5.01 – Assegurar uma boa gestão para qualquer projeto em que trabalhem, incluindo procedimentos eficazes para promoção da qualidade e redução de risco.

5.02 – Certifique-se de que os engenheiros de *software* sejam informados sobre os padrões antes de serem obrigados a eles.

5.03 – Certifique-se de que os engenheiros de *software* conheçam as políticas e procedimentos do empregador para proteger senhas, arquivos e informações confidenciais para o empregador ou confidenciais para outros.

5.04 – Atribua trabalho somente depois de levar em consideração as contribuições apropriadas de educação e experiência, temperadas com o desejo de promover essa educação e experiência.

5.05 – Garanta estimativas quantitativas realistas de custo, cronograma, pessoal, qualidade e resultados em qualquer projeto em que trabalhem ou se proponham a trabalhar e forneçam uma avaliação da incerteza dessas estimativas.

5.06 – Atraia potenciais engenheiros de *software* apenas pela descrição completa e precisa das condições de emprego.

5.07 – Oferecer remuneração justa.

5.08 – Não impedir injustamente que alguém assuma um cargo para o qual essa pessoa esteja devidamente qualificada.

5.09 – Certifique-se de que haja um acordo justo sobre a propriedade de qualquer *software*, processo, pesquisa, escrita ou outra propriedade intelectual para a qual um engenheiro de *software* tenha contribuído.

5.10 – Providenciar o devido processo legal ao ouvir acusações de violação da política de um empregador ou deste Código.

5.11 – Não peça a um engenheiro de *software* para fazer qualquer coisa inconsistente com este Código.

5.12 – Não punir ninguém por expressar preocupações éticas sobre um projeto.

Princípio 6 – Profissão

Os engenheiros de *software* devem promover a integridade e a reputação da profissão de acordo com o interesse público. Em particular, os engenheiros de *software* devem, conforme apropriado:

6.01 – Ajudar a desenvolver um ambiente organizacional favorável à atuação ética.

6.02 – Promover o conhecimento público da engenharia de *software*.

6.03 – Amplie o conhecimento de engenharia de *software* por meio da participação adequada em organizações profissionais, reuniões e publicações.

6.04 – Apoiar, como membros de uma profissão, outros engenheiros de *software* que se esforçam para seguir este Código.

6.05 – Não promover seu próprio interesse em detrimento da profissão, cliente ou empregador.

6.06 – Obedecer a todas as leis que regem seu trabalho, a menos que, em circunstâncias excepcionais, tal cumprimento seja inconsistente com o interesse público.

6.07 – Seja preciso ao declarar as características do *software* no qual eles trabalham, evitando não apenas alegações falsas, mas também alegações que possam ser consideradas especulativas, vazias, enganosas ou duvidosas.

6.08 – Assuma a responsabilidade de detectar, corrigir e relatar erros no *software* e documentos associados nos quais eles trabalham.

6.09 – Certifique-se de que clientes, empregadores e supervisores conheçam o compromisso do engenheiro de *software* com este Código de ética e as ramificações subsequentes de tal compromisso.

6.10 – Evite associações com empresas e organizações que estejam em conflito com este Código.

6.11 – Reconheça que as violações deste Código são inconsistentes com ser um engenheiro de *software* profissional.

6.12 – Expresse preocupação às pessoas envolvidas quando forem detectadas violações significativas deste Código, a menos que isso seja impossível, contraproducente ou perigoso.

6.13 – Denuncie violações significativas deste Código às autoridades competentes quando estiver claro que a consulta às pessoas envolvidas nessas violações significativas é impossível, contraproducente ou perigosa.

Princípio 7 – Colegas

Os engenheiros de *software* devem ser justos e apoiar seus colegas. Em particular, os engenheiros de *software* devem, conforme apropriado:

7.01 – Incentive os colegas a aderirem a este Código.

7.02 – Auxiliar os colegas no desenvolvimento profissional.

7.03 – Dê crédito total ao trabalho de outros e evite receber crédito indevido.

7.04 – Revise o trabalho de outras pessoas de maneira objetiva, sincera e devidamente documentada.

7.05 – Dar uma audiência justa às opiniões, preocupações ou reclamações de um colega.

7.06 – Ajude os colegas a estarem totalmente cientes das práticas de trabalho padrão atuais, incluindo políticas e procedimentos para proteger senhas, arquivos e outras informações confidenciais e medidas de segurança em geral.

7.07 – Não intervir injustamente na carreira de qualquer colega; no entanto, a preocupação com o empregador, o cliente ou o interesse público pode obrigar os engenheiros de *software*, de boa-fé, a questionar a competência de um colega.

7.08 – Em situações fora de sua própria área de competência, peça a opinião de outros profissionais que tenham competência nessa área.

Princípio 8 – Eu

Os engenheiros de *software* devem participar da aprendizagem ao longo da vida sobre a prática de sua profissão e devem promover uma abordagem ética na prática da profissão. Em particular, os engenheiros de *software* devem se esforçar continuamente para:

8.01 – Aprofundar seus conhecimentos de desenvolvimentos na análise, especificação, projeto, desenvolvimento, manutenção e teste de *software* e documentos relacionados, juntamente com o gerenciamento do processo de desenvolvimento.

8.02 – Melhorar sua capacidade de criar *softwares* de qualidade seguro, confiável e útil a um custo razoável e dentro de um prazo razoável.

8.03 – Melhore sua capacidade de produzir documentação precisa, informativa e bem escrita.

8.04 – Melhorar sua compreensão do *software* e documentos relacionados em que trabalham e do ambiente em que serão usados.

8.05 – Melhorar o seu conhecimento das normas relevantes e da lei que rege o *software* e documentos relacionados em que trabalham.

8.06 – Aprimorar seu conhecimento deste Código, sua interpretação e sua aplicação ao seu trabalho.

8.07 – Não dar tratamento injusto a ninguém por causa de preconceitos irrelevantes.

8.08 – Não influenciar outros a realizar qualquer ação que envolva uma violação deste Código.

8.09 – Reconheça que as violações pessoais deste Código são inconsistentes com ser um engenheiro de *software* profissional."

Atuação Profissional

Atualmente, a engenharia de *software* e a atuação do engenheiro de *software* correspondem menos a codificação, programação e linguagens, apesar de ainda serem dominantes e muito mais sobre paradigmas de desenvolvimento, estilos, formalização, componentes reutilizáveis, bibliotecas, controles de versionamento, entrega contínua e, claro, escrita de artefatos de *softwares* mais eficientes.

Porém, entenda que as linguagens (dezenas e dezenas delas) são apenas ferramentas que você deve conhecer. Você não precisa dominar várias linguagens; o que você precisa saber são

conceitos de programação e os “pilares” de engenharia e desenvolvimento de *software*.

Assim, alguns conceitos que devem ser dominados nesta Unidade são os seguintes:

- Ciclo de vida de desenvolvimento de *software*;
- Complexidade/eficiência de tempo de execução e espaço;
- Estruturas de dados e algoritmos: *hashmaps*, listas vinculadas, pilhas, gráficos, entre outros;
- Programação dinâmica;
- Programação orientada a objetos/programação funcional;
- Lógica;
- Teste;
- Sistemas operacionais;
- Matemática.

O mais importante é que os engenheiros de *software* estejam preparados para desenvolver *softwares* eficientes e de alta qualidade a fim de os implementar em uma linguagem sensata para aquele contexto. Tudo o que foi mencionado até então pode ser implementado em todos os idiomas, de modo que você realmente precisará trabalhar com uma linguagem ou com um idioma específico, especialmente se estiver participando de um projeto com esse idioma/linguagem em uso, ou caso simplesmente faça mais sentido usar sempre a mesma linguagem.

Por outro lado, entende-se de forma geral porque todos veem a engenharia de *software* e, por conseguinte, o serviço de um engenheiro de *software* como o de um desenvolvedor de *software*.

Isso se deve comumente porque eles criam *softwares* para computadores e aplicativos. Se você for um(a) pensador(a) analítico(a) que gosta de resolver problemas e tornar os produtos digitais mais fáceis de se usar, a carreira de engenharia de *software* poderá ser recompensadora. Contudo, nunca tire do horizonte as seguintes responsabilidades mínimas e que vão muito além da simples programação/desenvolvimento:

- Projetar e manter sistemas de *software*;
- Avaliação e teste de novos programas de *software*;
- Otimização de *software* para velocidade e escalabilidade;
- Escrita e teste de código;
- Ações de consultoria com clientes, engenheiros, especialistas em segurança e outras partes interessadas em um projeto de *software*;
- Apresentação de novos recursos para as partes interessadas e clientes internos;
- Garantir que diferentes tipos de programas de *software* se comuniquem entre si em uma plataforma;
- Criar e aplicar padrões de TI em uma infraestrutura;
- Manter a documentação dos sistemas de TI;
- Atualizar-se para novas tecnologias, conforme necessário;
- Colaborar com equipes de desenvolvimento, arquitetos de sistemas sênior e profissionais de Ciência de Dados;
- Desenvolver aplicativos para iOS, *Android*, *Windows* ou outros sistemas operacionais;
- Realizar uma análise de requisitos e ajustes de *software*, conforme necessário;

- Liberar atualizações de *software*;
- Trabalhar com *designers* gráficos, equipe de atendimento ao cliente, gerentes de projeto e outros departamentos voltados para o cliente;
- Arquitetura e modelagem de banco de dados;
- Gerenciamento de projetos *waterfall*, híbrido e ágil – com *Scrum*, pelo menos;
- Conhecer fundamentos de computação em nuvem;
- Controlar A versão de *software* usando suítes como GIT;
- Teste e depuração de *design*.

Caso você se interesse ou se importe com certificações e habilidades referentes à engenharia de *software* que lhe deem certa proficiência e que ajudem a construir reputação no mercado, confira as seguintes:

- Profissional Certificado de Desenvolvimento de *Software* (CSDP);
- Engenheiro de *software* certificado;
- C *Certified Professional Programmer* (*CLP*);
- Programador Profissional Certificado C++ (CPP);
- Ao menos 5 anos de experiência em desenvolvimento de *software* ou programação;
- Ampla experiência em *Java*, *Python*, C++ e outras linguagens de programação;
- Conhecimento profundo em HTML5, CSS3 e CMS;

- Conhecimento prático de SQL enquanto desenvolvedor certificado pela AWS.

Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta Unidade:

Vídeo

O que um Engenheiro de *Software* Faz?

Leitura

Os Maiores Desafios e as Barreiras no Desenvolvimento de *Software*

Clique no botão para conferir o conteúdo.

ACESSE

Quatro Tendências que Moldar a Engenharia de *Software* em 2020

Clique no botão para conferir o conteúdo.

ACESSE

Engenharia de *Software*: o que é, como se Tornar e Conselhos de quem Está na Área

Clique no botão para conferir o conteúdo.

ACESSE

Referências

ASSOCIATION FOR COMPUTER MACHINERY et al. Software engineering code of ethics. ***Commun.*** **ACM**, v. 40, n. 11, p. 110-118, 1997. Disponível em: <https://ethics.acm.org/code-of-ethics/software-engineering-code>. Acesso em: 11/06/2022.

CAMBRIDGE dictionary. 2022. Disponível em: <https://dictionary.cambridge.org/dictionary/english/engineer>. Acesso em: 13/07/2022.

DAVIS, M. *Will software engineering ever be engineering*? ***Communications of the*** **ACM**, v. 54 n. 11, p. 32-34, 2011. Disponível em: <https://cacm.acm.org/magazines/2011/11/138199-will-software-engineering-ever-be-engineering/fulltext>. Acesso em: 13/07/2022.

MARTIN, M. *What is software engineering? Definition*, *basics*, *characteristics*. **Guru99**, 2022. Disponível em: <https://www.guru99.com/what-is-software-engineering.html>. Acesso em: 13/07/2022.

PRESSMAN, R. S. **Engenharia de *software*:** uma abordagem profissional. 8. ed. São Paulo: McGraw-Hill, 2016.